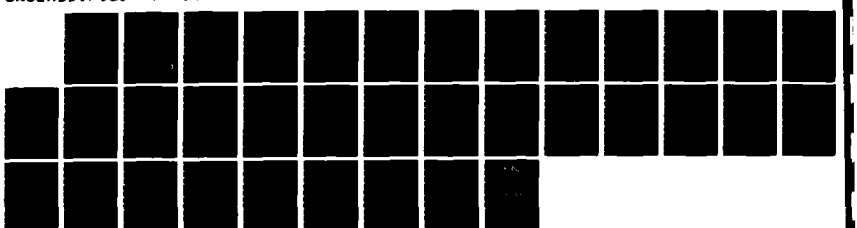
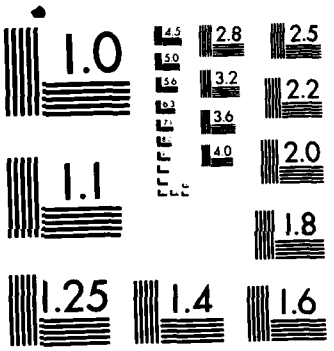
AD-A161 930 FUNCTIONAL TESTING OF LSI/VLSI BASED SYSTEMS WITH MEASURE OF FAULT COVERAGE(U) STATE UNIV OF NEW YORK AT ALBANY RESEARCH FOUNDATION S Y SU ET AL 0 NOV 83 1/1

UNCLASSIFIED DAAB07-82-K-J056 F/G 9/2 NL





SCIENTIFIC AND TECHNICAL REPORT

FIFTH QUARTERLY STATUS REPORT

Prepared By

Stephen Y.H. Su
Project Director
Dept. of Computer Science
State University of New York
Binghamton, New York 13901
(607)-798-2296 (office)
(607)-798-4802 (secretary)

A. Contractor's name and address:

The Research Foundation of
State University of New York
P.O. Box 9, Albany, New York 12201

B. Contract No. DAAB07-82-K J05-6

C. Date of Report - November 3, 1983

D. Title: The ~~Fourth~~ Quarterly Status Report for the project "Functional Testing of LSI/VLSI Based Systems with Measure of Fault Coverage"

E. Period Covered: July 28 to October 27, 1983

F. [illegible] of Progress: See attached

G. Spending: Actual Quarter Cost: $ 41,837.00

Cumulative Cost to date: $132,627.15

A FUNCTIONAL TESTING METHOD FOR MICROPROCESSORS*

by

Li Shen**

Stephen Y.H. Su

Department of Computer Science
Thomas J. Watson School of Engineering,
Applied Science, and Technology
State University of New York
Binghamton, N.Y. 13901

ABSTRACT

This paper presents a method for functional testing of microprocessors. First, we develop a control fault model at the RTL (Register Transfer Language) level. Based on this model, we establish testing requirements for control faults. We present three test procedures to verify the write and read sequences and use the write and read sequences to test other instructions in a microprocessor. By utilizing k-out-of-m codes, we can use fewer tests to cover more faults, thereby reducing the test generation time.

* This work is supported by the U.S. Army Communication Electronics Command under Research Contract No. DAAB 07-82-K-J056.

** Li Shen is now a visiting scholar in SUNY-Binghamton, from Institute of Computing Technology, Chinese Academy of Sciences, Beijing, China.

I. INTRODUCTION

The development of integrated circuit technology has resulted in a wide range of applications for microprocessors. Testing of microprocessors is a difficult problem because of the complexities of microprocessors. The problem is more serious for users due to lack of information on internal implementation of microprocessors and other VLSI chips. Over the past few years, several methods have been proposed to solve this problem. These testing techniques are essentially based on functional level [1-11].

A microprocessor is a type of complex sequential machine. The current approach is to test microprocessors by instruction execution. Generally, before executing an instruction-under-test we have to write certain data into some registers, and after executing the instruction, read the contents of the registers. Therefore, if the write or the read instruction is faulty, we may not be able to test the instruction-under-test. To solve this problem, Thatte and Abraham [3] have to label instructions and define test order in detail before testing. However, they do not consider the partial execution of an instruction. So for instruction decoding fault $I_j/I_j + I_k$, it is assumed that instead of executing I_j, both instructions I_j and I_k are executed to completion. It is more general and practical to consider partial execution of an instruction under fault. Our fault model allows this.

Abraham and Parker [6] proposed a simplified fault model. First, one tests all internal registers, then executes all instruction and data manipulation functions.

In this paper, we consider the basic instructions for the write and read register functions as the kernel of a microprocessor. This kernel can be represented by a sequential machine. Based on the fault model, we use checking experiment

to verify the kernel. Then we use the kernel for testing other instructions. The control fault model is established at the Register Transfer Language (RTL) level, since it is convenient to represent the instruction decoding faults and other control faults at such a level.

Section II presents a fault model for microprocessors, emphasizing the control fault model defined at the RTL level instead of the instruction level. In Section III, after examining most existing off-the-shelf microprocessors, we derive testing requirements based on different types of operations. In Section IV, we define the write and read sequences as the kernel of a microprocessor. Then Section V presents the verification of the write and read sequences. Section VI discusses the testing of control faults. Finally, conclusions are given in Section VII.

II. FAULT MODEL

The functions of a microprocessor are mainly performed by instruction execution. The sequence of operations for an instruction can be described by RTL. We consider that an instruction consists of a series of RTL statements. The typical statement is defined as

(conditions): $D \leftarrow f(S_1, S_2, \ldots, S_i, \ldots)$

where

D - destination

S_i - Source

$f(S_1, S_2, \ldots, S_i, \ldots)$ - operation

Destinations and sources may be internal registers of a microprocessor or external to the microprocessor (i.e., data bus, address bus, etc.). We are only concerned with those internal registers which are of interest to users, so we do not consider implied registers such as buffers. For example, data transfer from memory to memory can be described as $DB_j \leftarrow DB_i$, instead of $Buffer \leftarrow DB_i$ followed by $DB_j \leftarrow Buffer$, where DB denotes the data bus which represents data input or output of memory, i,j denote different bus cycles, DB_i (read from memory) is ahead of DB_j (write into memory).

After examining most existing off-the-shelf microprocessors, e.g. Intel 8080 and 8086, Zilog 80 and 8000, Motorola 6800 and 68000, the RTL-like operations can be divided into two classes, transfer operations (class T, $D \leftarrow S$), and arithmetic and logical operations (class A). Class A can be subdivided into six subclasses based on the combination of destinctions and sources as shown in Table 1, where the content of flag bits constitute a status register.

Class	Type of Expression	Operation
A1	$D \leftarrow f(D)$	BIT SET BIT RESET BIT COMPLEMENT INCREMENT DECREMENT DECIMAL ADJUST SHIFT ROTATE COMPLEMENT NEGATE CLEAR
A2	$D \leftarrow f(D,S)$	ADDITION ADDITION WITH CARRY SUBTRACTION SUBTRACTION WITH BORROW AND OR XOR
	$D \leftarrow f(S)$	EXTEND SIGN
A3	$D \leftarrow f(S_1,S_2)$	ADDITION, $D \leftarrow S_1+S_2$
	$D \leftarrow f(D,S_1,S_2)$	ADDITION, $D \leftarrow D+S_1+S_2$
A4	$D \leftarrow f(S_1,S_2,S_3)$	ADDITION, $D \leftarrow S_1+S_2+S_3$
AM	$D_1,D_2 \leftarrow f(D_1,S)$	MULTIPLY DIVIDE
AF	$Flags \leftarrow f(S)$	BIT TEST
	$Flags \leftarrow f(S_1,S_2)$	COMPARE
	$Flags \leftarrow f(S_1,S_2,\ldots)$	Modifying flags for all arithmetic and logical instructions

Table 1. RTL-Like Operations

Note that the control operations in RTL control statements such as conditional branch are not listed because we can use RTL assignment statements with conditions and expand the RLT description for instructions with loops.

A microprocessor usually can be divided into two sections: the data processing part and the control part [2,11]. In this paper, we consider faults in both parts with the emphasis on the control faults.

A. Data processing faults

(1) Data storage fault (R)/(R)'

This means that the content of register is changed from (R) to (R)' due to faults such as stuck-at and pattern sensitive faults.

(2) Data transfer fault ←/←'

The fault occurs in the transfer path between the sources and the destination. This type of fault includes stuck-at, bridging and pattern sensitive faults.

(3) Data manipulation fault (f)/(f)'

This is the operation execution fault. Under this fault, the operation f is executed, but the result of operation is wrong.

B. Control faults

This kind of fault involves register decoding faults, instruction decoding faults and other control faults. A register decoding fault means missing or changing the selected register, or selection of an extra register, denoted by R/ϕ, R/R', and R/R+R' respectively. For instruction decoding faults, we consider that an instruction can be executed partially. It means missing or changing the selected operation, or selection of an extra operation in RTL. In this case, the instruction decoding fault may be I_j/ϕ, $I_j/\Delta I_j$, $I_j/\Delta I_k$, I_j/I_k, $I_j/\Delta I_j+\Delta I_k$, $I_j/I_j+\Delta I_k$, I_j/I_j+I_k, and so forth, where ΔI means part of instruction I.

In fact, for most existing off-the-shelf microprocessors, the register decoding faults can be considered as instruction decoding faults, i.e. $R/\phi \varepsilon I_j/\phi$, $R/R' \varepsilon I_j/I_k$, $R/R+R' \varepsilon I_j/I_j+I_k$, where I_j, I_k are transfer instructions among registers. For example, let I_j be $R_j \leftarrow$memory, I_k be $R_k \leftarrow$memory, then fault R_j/R_k εfault I_j/I_k. In addition, the control faults also include instruction execution sequence faults, condition faults and so on.

From the above observation, we assert that it is appropriate to represent the control faults at the RTL level. Therefore, we will define the above control faults at such a level. Let f denote $D \leftarrow f(S_1, S_2, \ldots)$, which is an operation on the instruction-under-test, and $f \varepsilon \{f\}$, where $\{f\}$ is the set of RTL operations of a microprocessor. Let f' denote $D' \leftarrow f'(S_1', S_2', \ldots)$, which is an unexpected (faulty) operation, and $f' \varepsilon \{f\}$.

We now define three classes (i.e. nine subclasses F1, F2, ..., F9) of control faults as follows:

(1) f/ϕ- No operation is executed.

F1. $\underline{f/\phi}$

(2) f/f' - Instead of performing the operation f, another operation f' is executed. It contains two subclasses of faults.

F2. $\underline{\delta f/f'}$: Here δ means that the destination registers D and D' are different and the fault is f/f'.

F3. $\underline{\sigma f/f'}$: σ denotes that registers D and D' are the same.

(3) f/f+f' - In addition to operation f, another operation f' is also executed. It can be subdivided as follows.

(3a) Register D and D' are different.

F4. $\underline{\delta f/f+f'}$: The source register list of f and f' does not include D' and D respectively. We are not concerned with the execution order

of f and f'.

F5. δf/f'f: The source register list of f includes D'. f' is executed before performing operation f; i.e. the execution order is

1. $D' \leftarrow f'\ (S_1', S_2', \ldots)$
2. $D \leftarrow f\ (S_1, S_2, \ldots, D'^*)$

where register without * denotes its content before executing the operation, register with * denotes its content after executing the operation.

F6. δf/ff': The source register list of f' includes D and the execution order is

1. $D \leftarrow f\ (S_1, S_2, \ldots)$
2. $D' \leftarrow f'\ (S_1', S_2', \ldots, D^*)$

(3b) Registers D and D' are the same, i.e. D'=D. When the source register list of f and f' does not include D' and D respectively, if the execution order is f'f, the fault does not affect the execution of f. If the execution order is f f', it is the same as the case with the fault σf/f'.

F7. σf/f'f: The source register list of f includes D, and the execution order is

1. $D \leftarrow f'\ (S_1', S_2', \ldots)$
2. $D \leftarrow f\ (S_1, S_2, \ldots, D^*)$

F8. σf/ff': The source register list of f' includes D and the execution order is

1. $D \leftarrow f\ (S_1, S_2, \ldots)$
2. $D \leftarrow f'\ (S_1', S_2', \ldots, D^*)$

F9. σf/fLf': Both f and f' are executed at the same time.

$D \leftarrow f\ (S_1, S_2, \ldots)$

$D \leftarrow f'\ (S_1', S_2', \ldots)$

where L denotes logical AND or OR function. In this case, the final content of the destination D is the result of the composite value (i.e. the result of the

AND or OR function) of f and f*.

Note that the above control faults can occur at any place in an instruction execution sequence. This control fault model can cover register decoding faults, instruction decoding faults (including partially instruction execution), instruction execution sequence faults, etc., since any control fault can always be defined as missing, changing, or extra RTL opeations and will cause registers to have wrong contents.

III. REQUIREMENTS FOR TESTING CONTROL FAULTS

Our purpose is to test the execution of microprocessor instructions. Therefore, the objective of test pattern generation is to find the initial data in registers (test data) needed for testing functions of an instruction and also the sequence of instructions. This test data must satisfy certain requirements. From the control fault model given in Section II, we can obtain various requirements for testing control faults.

Let us establish the following notation. For a fault-free operation f, we have

V_i = the value of register i.

VS_i = the value of the operand in source register S_i.

VD = the value of the operand in destination register D.

VD* = the value of destination register D storing the result of operation f.

For a faulty operation f', we obtain VS_i', VD' and VD'* instead.

Theorem 1. Control faults T/ϕ, T/T' and T/T+T' can be detected if the data values of registers satisfy the following requirements:

QTT1. $V_i \neq V_j$, $i \neq j$

QTT2. $V_i \; L \; V_j \neq V_i$, $i \neq j$

Proof. We shall prove this theorem by considering the nine fault classes defined in Section II.

(i) For fault F1 (T/ϕ) and F2 (δT/T'), in order to verify transfer operation T, one needs VS $\neq$ VD.

(ii) For fault F3 (σT/T'), the results of T and T' should be different, i.e. VD* $\neq$ VD'*. To obtain this result, we must have VS $\neq$ VS'.

(iii) For fault F4 (δT/T+T') and F5 (δT/T'T), we need only to detect the extra operation T'. Therefore, VS' $\neq$ VD'.

(iv) Fault F6 ($\delta T/TT'$) means that transfer operation D←S is performed first, then D'←D*. In order to detect the extra operation T', one needs $VS \neq VD'$.

(v) Fault F7 ($\sigma T/T'T$) yields D←S' followed by D←D*. Therefore, we obtain the requirement $VS' \neq VD$.

(vi) For fault F8 ($\sigma T/TT'$), we have D←S, then D←D*. This fault does not affect operation T. In order to verify T, we need $VS \neq VD$.

The above six requirements belong to QTT1.

(vii) For fault F9 ($\sigma T/TLT'$), the composite value of both results of T and T' should be different from the correct result of T. i.e. $VSLVS' \neq VS$ which belongs to QTT2.

Q.E.D.

Theorem 2. Control faults T/A' and T/T+A' can be detected if the data values of registers satisfy the following requirements:

QTA1. $V_i \neq V_j$, $i \neq j$

QTA2. $f'_A \neq VS$

QTA3. $f'_A \neq VD'$

QTA4. $VSLf'_A \neq VS$

where f'_A is the result of operation of class A, i.e. $f'_A = f'_A(VS'_1, VS'_2, \ldots)$.

Proof. The proof is similar to the proof for Theorem 1. Since A' instead of T' is performed, we can change VS' to f'_A in the requirements, (ii), (iii), (v) and (vii) in the Proof for Theorem 1 to obtain the corresponding requirements for Theorem 2.

(i) For F2 ($\delta T/A'$), $VS \neq VD$, (QTA1).

(ii) For F3 ($\sigma T/A'$), $VS \neq f'_A$, (QTA2).

(iii) For F4 ($\delta T/T+A'$) and F5 ($\delta T/A'T$), $f'_A \neq VD'$, (QTA3).

(iv) For F6 ($\delta T/TA'$), it means that D←S first, then $D' \leftarrow f'_A(S'_1, S'_2, \ldots, D^*)$.

We need f'_A $(VS'_1, VS'_2, \ldots, VS) \neq VD'$. Since VS can be selected as any initial data value, it can be considered as one of several source operands. Therefore, we can rewrite f'_A $(VS'_1, VS'_2, \ldots, VS) \neq VD'$ as $f'_A \neq VD'$, (QTA3).

(v) For F7 ($\sigma T/A'T$), $f'_A \neq VD'$, (QTA3).

(vi) For F8 ($\sigma T/TA'$), it implies D←S followed by D←f'_A $(S'_1, S'_2, \ldots, D^*)$. This requires $VS \neq f'_A$ $(VS', VS', \ldots, VS)$. Here VS can be considered as a destination operand VD'. So we rewrite the inequality as $VD' \neq f'_A$, (QTA3).

(vii) For F9 ($\sigma T/TLA'$), $VS L f'_A = VS$, (QTA4). Q.E.D.

Theorem 3. Control faults A/ϕ, A/T' and A/A+T' can be detected if the data valued of registers satisfy the following inequalities.

QAT1. $f_A \neq VD$

QAT2. $f_A \neq VS'$

QAT3. $V_i \neq V_j$, $i \neq j$

QAT4. $f_A \neq VD'$

QAT5. f_A $(VS') \neq f_A(VD)$

QAT6. f_A L $VS' \neq f_A$

where $f_A = f_A$ $(VS_1, VS_2, \ldots)$, $f_A(VD) = f_A(VS_1, VS_2, \ldots, VD)$, $f_A(VS') =$ f_A $(VS_1, VS_2, \ldots, VS')$.

Proof. Since arithmatical and logical operations insteed of transfer operations are considered here, we can change VS to f_A in the cases (i), (ii), (iv), (vi) and (vii) of Theorem 1.

(i) For F1 and F2, $f_A \neq VD$, (QAT1).

(ii) For F3, $f_A \neq VS'$, (QAT2).

(iii) For F4 and F5, $VS' \neq VD'$, (QAT3).

(iv) For F6, $F_A \neq VD'$, (QAT4).

(v) F7 ($\sigma A/T'A$) means that D←S' followed by D←$f_A(S_1, S_2, \ldots, D^*)$ which yields $f_A(VS_1, VS_2, \ldots, VS')$. When there is no fault, $f_A(VS_1, VS_2, \ldots, VD)$ is obtained.

Thus the condition for detecting this fault is $f_A(VS_1, VS_2,\ldots,VS') \neq f_A(VS_1,VS_2,\ldots,VD)$, (QTA5).

(vi) For F8, $f_A \neq VD$, (QTA1).

(vii) For F9, $f_A \; L \; VS' \neq f_A$, (QTA6). Q.E.D.

Theorem 4. Control faults A/A' and A/A+A' can be detected if the data values of registers satisfy the following inequalities.

QAA1. $f_A \neq VD$

QAA2. $f_A \neq f'_A$

QAA3. $f'_A \neq VD'$

QAA4. $f'_A(f_A) \neq VD'$

QAA5. $f_A\ (f'_A) \neq f_A(VD)$

QAA6. $f'_A\ (f_A) \neq f_A$

QAA7. $f_A \; L \; f'_A \neq f_A$

where $f'_A(f_A) = f'_A(VS'_1, VS'_2,\ldots,f_A)$, $f_A(f'_A) = f_A(VS_1,VS_2,\ldots,f'_A)$.

Proof. For the same reason, we may change VS to f_A, and VS' to f'_A for cases (i) to (iii) and (vii) in Theorem 1 to obtain QAA1 to QAA3 and QAA7 respectively. In addition, since the results of A and A' may affect each other, we can obtain QAA4 to QAA6. Q.E.D.

Note that for requirement QAA7, if operation f_A and f'_A of class A are executed in the same unit (e.g. ALU), then both results of f_A and f'_A can no longer be considered as obtained separately. Instead, QAA7 may be considered as a data manipulation fault $(f_A)/(f_A)'$.

IV. WRITE AND READ SEQUENCES

As a microprocessor is one type of sequential machine and all internal registers are memory elements of the sequential machine, the content of registers represents the state of the sequential machine. Therefore, the following procedure is utilized for testing microprocessors.

1. Initialization of state of registers.
2. Execution of the instruction-under-test.
3. Read the state of registers.

In fact, Steps 1 and 3 consist of write and read register sequences respectively. Obviously, if we can guarantee the correctness of Steps 1 and 3 first, then the testing problem will be simplified.

The testing approach used here is a kind of open loop testing [6]. It implies the use of a test equipment which provides the stimuli to the microprocessor and observes the responses from the microprocessor.

Now let us discuss write and read sequences which are used for writing and reading the register states of a microprocessor. They consist of several basic instructions, called the kernel of microprocessor. These instructions of the kernel can be carried out by a sequential machine, therefore, we can use a checking experiment to verify the kernel.

A. The kernel of microprocessor

Definition 1. Kernel instruction set - A small subset of instructions of a microprocessor which can be used for constituting the write and read register sequences.

Definition 2. Register set - All internal registers of a microprocessor from the view of the architecture or programming.

Definition 3. Kernel state - The register state, i.e. certain set of data values of the registers.

Definition 4. Kernel input - The write sequence for writing a set of data into the registers, or the read sequence for reading out the contents of the registers.

Definition 5. Kernel output - The set of data values of the registers which are read out by the read sequence.

B. Kernel instruction set

There exists many choices for the kernel instruction set. In order to keep the kernel small, the following requirements should be satisfied.

1. The number of instructions in the kernel instruction set should be small.

2. Functions of each kernel instruction should be as simple as possible. For instance, a kernel instruction contains mainly transfer type of operations, or small number of RTL operations.

3. In order to simplify addressing, the priority order of choosing the addressing mode of an instruction is as follows.

- For write register instructions: Immediate, Direct, Indirect.
- For read register instructions: Direct, Indirect.

For the existing off-the-shelf microprocessors, most registers can be written into or read from directly. These registers are called direct access registers. Others are indirect access registers which can be accessed through the direct access registers in certain order by using transfer instruction among registers.

C. Kernel state

In order to simplify the testing, during the checking experiment, we only use a few states for the good kernel, i.e. we define several sets of data value for the registers. Therefore, we should choose the data values (test data) such that they can cover as many faults as possible.

V. VERIFYING WRITE AND READ SEQUENCES

We use the checking experiment to verify the kernel of a microprocessor. The main task is to decide how many states of the kernel and what test data we use.

Abraham and Parker [6] use the k-out-of-m codes for their "register read test" procedure, where m is the width of a code word (i.e. the length of register), and k is the number of 1's in the code word. As we will see, this type of code is powerful since it can be used as test data to cover most control faults by using fewer data. We will use the k-out-of-m codes for verifying the write and read sequences as well as for testing control faults. The k-out-of-m codes can also detect stuck-at type faults, but do not guarantee to cover all data processing faults.

To simplify the testing, we shall only use transfer operations of the kernel instructions in write and read sequences. Therefore, we only need to consider the requirements of Theorems 1 and 2.

A. QTT1, QTA1 and QTT2

The k-out-of-m codes used as test data can satisfy requirements QTT1, QTA1 and QTT2 ($V_i \neq V_j$ and $V_i \; L \; V_j \neq V_i$). This is because in k-out-of-m codes, all code words are distinct and the AND(OR) operation of any two code words will reduce (increase) the number of 1's in the code word, thereby different from both original code words.

B. QTA2

During the checking experiment for the kernel, there exists an input leaving the kernel state unchanged. Therefore, transfer operation (in write sequence) D←S implies VD = VS, then the requirement QTA2, $f'_A \neq VS$, becomes $f'_A \neq VD$. This is for detecting fault $\sigma T/A'$, here D and D' are the same register. Therefore,

QTA2 is changed to $f'_A \neq VD'$ which belongs to QTA3.

C. QTA3

The requirement QTA3, $f'_A(VS', VS', \ldots) \neq VD'$, is for detecting an extra operation. We list the restrictions of operands (test data) for detecting operation of class A in Table 2.

Extra Operations	Restrictions of Operands
BIT SET	①
BIT RESET	①
BIT COMPLEMENT	No
INCREMENT	No
DECREMENT	No
DECIMAL ADJUST	③
SHIFT	$\neq \underline{0}$ (all 0s), $\underline{1}$ (all 1s)
ROTATE	$\neq \underline{0}, \underline{1}$
COMPLEMENT	No
NEGATE	No
CLEAR	$\neq \underline{0}$
ADDITION	$\neq \underline{0}$
ADDITION WITH CARRY	$\neq \underline{0}$, -1
SUBTRACTION	$\neq \underline{0}$
SUBTRACTION WITH BORROW	$\neq \underline{0}$, -1
AND	k-out-of-m codes
OR	k-out-of-m codes
XOR	k-out-of-m codes
EXTEND SIGN	$\neq \underline{0}, \underline{1}$
ADDITION, $D \leftarrow S_1 + S_2$	the least significant bit LSB =1
ADDITION, $D \leftarrow D + S_1 + S_2$	$\neq \underline{0}$; $VS'_1 + VS'_2 \neq 0$
ADDITION, $D \leftarrow S_1 + S_2 + S_3$	④
MULTIPLY	$\neq \underline{0}$, 1
DIVIDE	$\neq \underline{0}$, 1
BIT TEST	②
COMPARE	②
Modifying flags	⑤

Table 2. Restrictions of operands for detecting faulty arithmetic or logical operations

① If we use two tests in which the source operands are complements of each other, then one of the tests can detect the faulty operation.

② These operations only set or reset the flags. We use two tests with the identical source operands and two sets of flags which are complements to each other. The faulty will change one of the flags.

③ The execution of DECIMAL ADJUST (to add certain values) depends on the value of source operand and the flags. We can use either method in case 1 or either method in case 2 to detect the operation.

④ Operation $D \leftarrow S_1 + S_2 + S_3$ is only used for memory address addition. Here D means external address bus. In this case, the unexpected operation does not affect the write and read registers. Hence it needs not to be considered for verifying the kernel.

⑤ During the checking experiment for the kernel, if we have considered the main operations in arithmetic and logical instructions as the unexpected operation, then we need not consider modifying flags which are auxiliary operations.

For other operations, the restrictions are obvious. For example, ADDITION WITH CARRY, $D' \leftarrow D' + S' + CARRY$, if CARRY = 0 with the restriction $VS' \neq 0$ or CARRY' = 1 with the restriction is $VS' \neq -1$, then $VD'^* \neq VD'$. Since we use k-out-of-m codes as operands, these restrictions can be satisfied. For operation $D' \leftarrow D' + S_1' + S_2'$, since the negative value of any k-out-of-m code word will not be any k-out-of-m code, i.e. $VS_1' + VS_2' \neq 0$; so $VD' + VS_1' + VS_2' \neq VD'$. For operation $D' \leftarrow S_1' + S_2'$, we can divide the k-out-of-m codes into two groups with different LSB (Least Significant Bit). These two groups complement each other. The group with the restriction LSB = 1 will be used for testing.

For operation ROTATE, the restriction is for the case of the odd number of shifted bits. Otherwise, we need other restriction (e.g. using subset of k-out-of-m codes as operands).

D. QTA4

The checking experiment does not guarantee requirement QTA4, VS L $F'_A \neq$ VS. For example, for the normal transfer operation T, D←S, with VS = 0101, VD = 0110 (using 2-out-of-4 codes), if there exists a fault $\sigma T/TLA'$ and the extra operation A' is INCREMENT, D←D+1, then f'_A = VD + 1 = 0111. If L is an AND function, then VS L f'_A = VS. Thus QTA4 cannot be satisfied. Therefore, we need another test procedure to remedy this. The remedy is to change the value of S to $\underline{1}$ or $\underline{0}$ depending upon L being AND or OR respectively. From the above example, we see that if L is an AND function, let VS = $\underline{1}$ then QTA4 becomes $f'_A \neq \underline{1}$ which can be satisfied. Similarly, if L is an OR function, let VS = $\underline{0}$ and QTA4 changes to $f'_A \neq \underline{0}$.

Now let us check this remedy method for all operations of class A. Note that we only change the value of S in operation D←S, and D' in operation A' can be substituted by D.

(i) For class A_1, D←f'_A(D). If we use k-out-of-m codes, the new requirement QTA4', $f'_A \neq \underline{1}$ ($\underline{0}$), depending on L, can be satisfied except the operation CLEAR with L being OR. But in this case, the result of f'_A is always $\underline{0}$ which does not affect the operation D←S.

(ii) For class A_2, D←f'_A(D,S'). No matter S' is the same register as S or not, QTA4' is true except the following case: when S' and S are the same register, then for QTA4', $f'_A \neq 1$ (f'_A is operation OR and L is AND) and $f'_A \neq 0$ (f'_A is operation AND and L is OR) cannot be met. But in this case, if S' and S are the same, VSLf'_A is always the same as VS, the fault does not affect D←S, i.e. VSL(VD$\vee$VS) = VS and VS$\vee$(VD$\wedge$VS) = VS for any operands D and S.

(iii) For the EXTEND SIGN operation, $D \leftarrow f'_A(S')$, the result of f'_A is $\underline{0}$ or $\underline{1}$. Similarly with case (i), either QTA4 can be met or the fault does not affect operation T.

(iv) For classes A3 and AF, QTA4 can be satisfied. For the same reason as QTA3, we need to consider neither modifying flags operation nor $D \leftarrow S_1 + S_2 + S_3$.

(v) For MULTIPLY and DIVIDE, since the execution period of these operations generally is longer than that of transfer operations, fault $\sigma f/fLf'$ cannot exist and we do not consider QTA4.

In summary, during the checking experiment we only need three sets of test data for internal registers. Let n be the number of the internal register, m be the length of registers. Suppose that m is even. Let $\underline{V}$ denote a complete set of k-out-of-m code words. Usually, $k = \frac{m}{2}$. $|\underline{V}| = \binom{m}{m/2}$ will be the maximum number of distinct codes. We divide them into two groups, $\underline{V}_0$ and $\underline{V}_1$ with different LSB. These two groups are complementary to each other. Note that for most microprocessors, m is even, and $|\underline{V}| > 2n$.

Let $\{\alpha\} = \{\alpha_1, \alpha_2, \ldots, \alpha_n\}$, $\{\overline{\alpha}\} = \overline{\alpha}_1, \overline{\alpha}_2, \ldots, \overline{\alpha}_n\}$, $\{\alpha\}$ and $\{\overline{\alpha}\}$ belongs to $\underline{V}_0$ and $\underline{V}_1$ respectively. We now construct four sets of data as follows.

Flag register	Other registers
α_1	$\alpha_2, \alpha_3, \ldots, \alpha_n$
$\overline{\alpha}_1$	$\overline{\alpha}_2, \overline{\alpha}_3, \ldots, \overline{\alpha}_n$
α_1	$\overline{\alpha}_2, \overline{\alpha}_3, \ldots, \overline{\alpha}_n$
α_1	$\alpha_2, \alpha_3, \ldots, \alpha_n$

In order to satisfy requirement QTA3, we can choose any three sets of data as the initial values of registers (test data). In fact, one needs a few more data as external bus inputs during testing. Therefore, the final number of code

words in $\{\alpha\}$ and $\{\bar{\alpha}\}$ is larger than n.

It should be pointed out that the program counter (PC) is easy to test. We can put a direct addressing branch instruction at the end of a write sequence. This instruction stores a particular value into PC, then the content of PC is checked at the beginning of the read sequence by observing the address bus.

Now we will derive the checking sequence for the kernel. As we mentioned before, we only use three sets of test data for the kernel, namely a, b and c. First of all, just like a sequential machine we have to obtain a flow table of the kernel. We consider two cases.

Case 1. For a microprocessor without indirect access registers, we obtain the following flow table.

	W_a	W_b	W_c	R
A	① A	④ B	⑦ C	⑩ A,a
B	② A	⑤ B	⑧ C	⑪ B,b
C	③ A	⑥ B	⑨ C	⑫ C,c

Case 2. For a microprocessor with indirect access registers, we obtain the following table.

	W_a	W_b	W_c	R
A	① A	⑦ B	⑬ C	⑲ A*,a
B	② A	⑧ B	⑭ C	⑳ B*,b
C	③ A	⑨ B	⑮ C	㉑ C*,c
A*	④ A	⑩ B	⑯ C	㉒ A*,a*
B*	⑤ A	⑪ B	⑰ C	㉓ B*,b*
C*	⑥ A	⑫ B	⑱ C	㉔ C*,c*

where W_a, W_b, W_c - write sequences for writing a, b and c respectively.

R - read sequence.

A, B, C, A*, B*, C* - Kernel states. A, B and C are the states after applying the corresponding write sequences W_a, W_b, W_c. A*, B* and C* are the states after applying read sequence R.

a, b, c, a*, b*, c* - Kernel output sequences produced by read sequence R for states A, B, C, A*, B*, C* respectively.

① denote the state transition i.

Since we only use three sets of test data, the number of states of the kernel is constant. It means that the above flow tables are independent of microprocessors. Therefore, we can easily obtain the checking sequences with the same form.

There are three requirements to derive a checking sequence [12,13].

1. Initialization of the machine (kernel) state using sychronizing sequence or homing sequence.

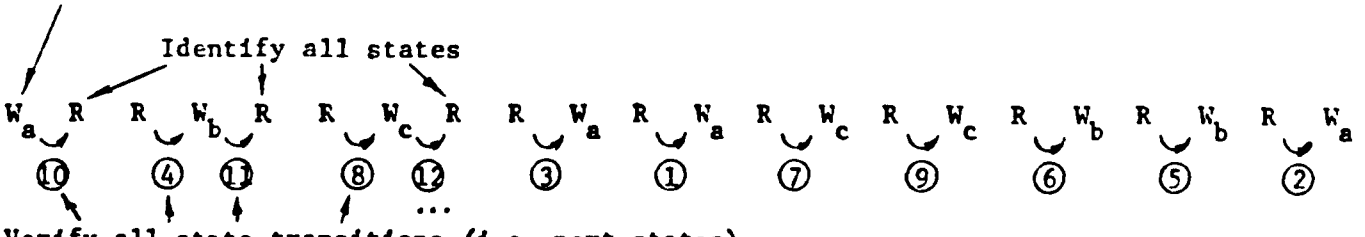
2. Identify all machine states using distinguishing sequence.

3. Verify each transition using distinguishing sequence.

For our kernel, there exists sychronizing sequences W_a or W_b or W_c and distinguishing sequence R. Therefore, we can easily derive checking sequences as follows.

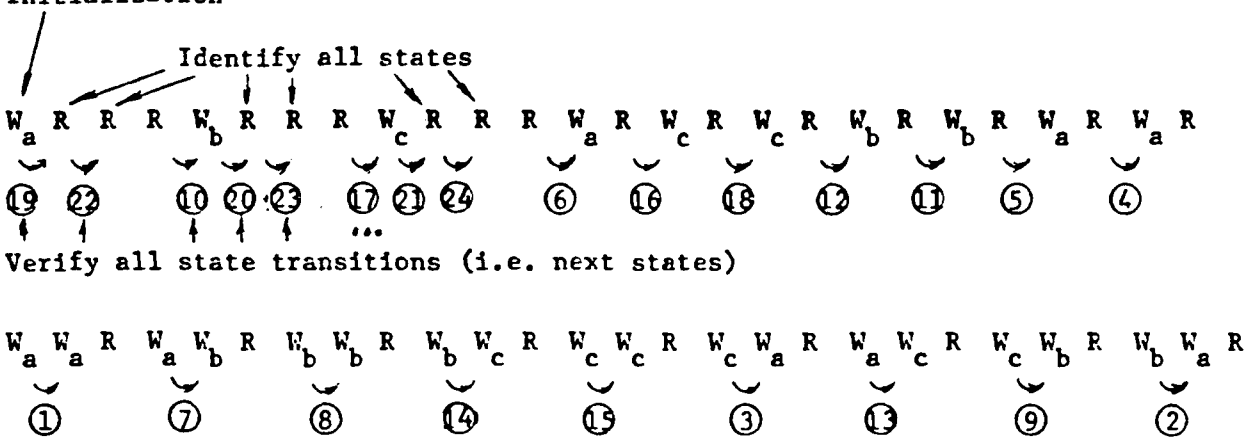
Checking Sequence 1 (for case 1)

Initialization



Verify all state transitions (i.e. next states).

Checking sequence 2 (for case 2)



Finally, we can obtain two test procedures for verifying the write and read sequences as follows.

Procedure 1: Checking experiment

1. If a microprocessor does not have indirect access registers, we use checking sequence 1.
2. If a microprocessor has indirect access registers, we use checking sequence 2.

Procedure 2: Remedy testing

For each of the three sets of register values a, b and c, do the following for each register.

1. Initialization of all registers.
2. Write 1 or 0 into the given register depending on the circuit implementation.
3. Read the given register.

Therefore, from the previous discussion in this section, we obtain the following theorem.

Theorem 5. Procedures 1 and 2 can verify write and read sequences, and after that the registers of a microprocessor can be initialized to any values.

VI. TESTING CONTROL FAULTS

During the verification of the correctness of write and read sequences, we are only concerned with certain transfer operations (not all RTL operations) in the kernel instructions. Therefore, when we test instruction decoding and other control faults, we need to test all instructions included in the kernel.

Procedure 3. Testing instruction decoding and other control faults

For each instruction, do the following steps,

1. Initialize register state using any particular initial values (test data).
2. Execute the instruction instruction-under test.
3. Read register state.

Note that we should first try to use three sets of data values a, b and c at Step 1.

Generally we need several tests for each instruction to detect the instruction decoding and other control faults. Obviously, the lower bound of the number of tests using Procedure 3 for each instruction is two. This is because any kind of microprocessors has several pairs of conditional branch instructions based on two different values for the same condition source. Therefore, when any instruction is under test, in order to detect an unexpected branch instruction due to a fault, we need at least two test patterns.

The upper bound on the test for each instruction is dependent upon the microprocessor-under-test. We can roughly estimate the order of tests for detecting instruction decoding faults. We consider n_I instructions to be tested, assume that each instruction corresponds to an operation, class T or class A, used for distinguishing instructions from each other. Let n_{IT} and n_{IA} denote the number of instructions which have operations class T and

class A respectively, i.e. $n_I = n_{IT} + n_{IA}$.

A. Testing instruction class T

1. The case of Theorem 1

Since for some condition branch instructions, their operations belong to class T, and they are condition transfer operations. In order to detect this unexpected operation T', two tests, in which test data are complement to each other, are sufficient.

2. The case of Theorem 2

First of all, we use three sets of initial values a, b and c for covering the requirements QTA1 and QTA3. In order to satisfy QTA2 and QTA4, we can modify a, b and c separately as new test data. As we have discussed in Section V, if the instruction-under-test has a transfer operation D←S, we can change VS in original data a, b and c to VD, then QTA2 becomes QTA3 which can be satisfied. Similarly, we can change VS to 1 (or 0) to satisfy QTA4. Here we need nine tests altogether. Thus, the order of the number of tests for testing instruction class T is $O(n_{IT})$.

B. Testing instruction class A

1. The case of Theorem 3

Test data a, b and c can cover QAT1 and QAT3. Similarly, in order to satisfy QAT2, QAT4, QAT5 and QAT6, we can modify a, b and c in turn. First, we change VS' and VD' to VD for covering QAT2 and QAT4 respectively. These changes are done for each register, so it needs 3n tests, where n is the number of registers. Then we change VS' to a particular value for covering QAT5 and QAT6 separately. It needs 6n tests. So the total number of tests for each instruction in this class will be 9n+3.

2. The case of Theorem 4

We need three tests using a, b and c for covering QAA1 and QAA2.

Then we attempt to find five particular tests to satisfy QAA3 through QAA7 for each unexpected operation A'. So the number of tests for each instruction in this class will be $3 + 5\ (n_{IA} - 1) = 5\ n_{IA} - 2$.

Therefore, the order of the number of tests for testing instruction class A is $0\ (n \cdot n_{IA} + n_{IA}^2)$.

VII. CONCLUSION

In this paper, we presented three test procedures to verify the write and read sequences and to test control faults at the RTL level.

A control fault model is defined at the RTL level instead of the instruction level. This allows us to model the fault more accurately.

Based on the control fault model, we consider the write and read register sequence functions as the kernel of a microprocessor independent of microprocessors. The similarity between the kernel of a microprocessor and a sequential machine allows us to obtain the checking sequences to verify the kernel. Therefore, testing of microprocessors can be simplified.

We mainly use k-out-of-m codes as test data which can cover more control faults. Therefore, the number of test patterns can be reduced.

Further work includes the enumeration of the control faults at the RTL level for the generation of tests for covering all possible control faults in a microprocessor.

ACKNOWLEDGEMENT

The authors thank Dr. K.K. Saluja for his helpful comments and suggestions on this paper.

VII. CONCLUSION

In this paper, we presented three test procedures to verify the write and read sequences and to test control faults at the RTL level.

A control fault model is defined at the RTL level instead of the instruction level. This allows us to model the fault more accurately.

Based on the control fault model, we consider the write and read register sequence functions as the kernel of a microprocessor independent of microprocessors. The similarity between the kernel of a microprocessor and a sequential machine allows us to obtain the checking sequences to verify the kernel. Therefore, testing of microprocessors can be simplified.

We mainly use k-out-of-m codes as test data which can cover more control faults. Therefore, the number of test patterns can be reduced.

Further work includes the enumeration of the control faults at the RTL level for the generation of tests for covering all possible control faults in a microprocessor.

ACKNOWLEDGEMENT

The authors thank Dr. K.K. Saluja for his helpful comments and suggestions on this paper.

VI. TESTING CONTROL FAULTS

During the verification of the correctness of write and read sequences, we are only concerned with certain transfer operations (not all RTL operations) in the kernel instructions. Therefore, when we test instruction decoding and other control faults, we need to test all instructions included in the kernel.

Procedure 3. Testing instruction decoding and other control faults

For each instruction, do the following steps,

1. Initialize register state using any particular initial values (test data).
2. Execute the instruction instruction-under test.
3. Read register state.

Note that we should first try to use three sets of data values a, b and c at Step 1.

Generally we need several tests for each instruction to detect the instruction decoding and other control faults. Obviously, the lower bound of the number of tests using Procedure 3 for each instruction is two. This is because any kind of microprocessors has several pairs of conditional branch instructions based on two different values for the same condition source. Therefore, when any instruction is under test, in order to detect an unexpected branch instruction due to a fault, we need at least two test patterns.

The upper bound on the test for each instruction is dependent upon the microprocessor-under-test. We can roughly estimate the order of tests for detecting instruction decoding faults. We consider n_I instructions to be tested, assume that each instruction corresponds to an operation, class T or class A, used for distinguishing instructions from each other. Let n_{IT} and n_{IA} denote the number of instructions which have operations class T and

class A respectively, i.e. $n_I = n_{IT} + n_{IA}$.

A. Testing instruction class T

1. The case of Theorem 1

Since for some condition branch instructions, their operations belong to class T, and they are condition transfer operations. In order to detect this unexpected operation T', two tests, in which test data are complement to each other, are sufficient.

2. The case of Theorem 2

First of all, we use three sets of initial values a, b and c for covering the requirements QTA1 and QTA3. In order to satisfy QTA2 and QTA4, we can modify a, b and c separately as new test data. As we have discussed in Section V, if the instruction-under-test has a transfer operation D←S, we can change VS in original data a, b and c to VD, then QTA2 becomes QTA3 which can be satisfied. Similarly, we can change VS to 1 (or 0) to satisfy QTA4. Here we need nine tests altogether. Thus, the order of the number of tests for testing instruction class T is $O(n_{IT})$.

B. Testing instruction class A

1. The case of Theorem 3

Test data a, b and c can cover QAT1 and QAT3. Similarly, in order to satisfy QAT2, QAT4, QAT5 and QAT6, we can modify a, b and c in turn. First, we change VS' and VD' to VD for covering QAT2 and QAT4 respectively. These changes are done for each register, so it needs 3n tests, where n is the number of registers. Then we change VS' to a particular value for covering QAT5 and QAT6 separately. It needs 6n tests. So the total number of tests for each instruction in this class will be 9n+3.

2. The case of Theorem 4

We need three tests using a, b and c for covering QAA1 and QAA2.

Then we attempt to find five particular tests to satisfy QAA3 through QAA7 for each unexpected operation A'. So the number of tests for each instruction in this class will be $3 + 5\ (n_{IA} - 1) = 5\ n_{IA} - 2$.

Therefore, the order of the number of tests for testing instruction class A is $0\ (n \cdot n_{IA} + n_{IA}^2)$.

VII. CONCLUSION

In this paper, we presented three test procedures to verify the write and read sequences and to test control faults at the RTL level.

A control fault model is defined at the RTL level instead of the instruction level. This allows us to model the fault more accurately.

Based on the control fault model, we consider the write and read register sequence functions as the kernel of a microprocessor independent of microprocessors. The similarity between the kernel of a microprocessor and a sequential machine allows us to obtain the checking sequences to verify the kernel. Therefore, testing of microprocessors can be simplified.

We mainly use k-out-of-m codes as test data which can cover more control faults. Therefore, the number of test patterns can be reduced.

Further work includes the enumeration of the control faults at the RTL level for the generation of tests for covering all possible control faults in a microprocessor.

ACKNOWLEDGEMENT

The authors thank Dr. K.K. Saluja for his helpful comments and suggestions on this paper.

REFERENCES

[1] A.C.L. Chiang and R. McCaskill, "Two New Approaches Simplify Testing of Microprocessors", Electronics, 22 Jan. 1976, p. 100.

[2] S.M. Thatte and J.A. Abraham, "Methodology for Functional Level Testing of Microprocessors", 8th International Symposium on Fault-Tolerant Computing, Toulouse, France, June 1978, pp. 90-95.

[3] S.M. Thatte and J.A. Abraham, "Test Generation for Microprocessors", IEEE Trans. on Computers, C-29, No. 6, June 1980, pp. 429-441.

[4] S.M. Thatte, "Test Generation for Microprocessors", Coordinated Science Laboratory Report R-842, University of Illinois at Urbana-Champaign, May 1979.

[5] J.A. Abraham and S.M. Thatte, "Fault Coverage of Testing Program for a Microprocessor", 1979 Test Conference, Oct. 1979, pp. 18-22.

[6] J.A. Abraham and K.P. Parker, "Practical Microprocessor Testing: Open and Closed Loop Approaches", IEEE Compcon, Spring 1981, pp. 308-311.

[7] Y. Min and S.Y.H. Su, "Testing Functional Faults in VLSI", 19th Design Automation Conference, Los Vegas, Nevada, 1982, pp. 384-392.

[8] K.K. Saluja, L. Shen and S.Y.H. Su, "A Simplified Algorithm for Testing Microprocessors", 1983 Test Conference, Oct. 1983, pp. 668-675.

[9] C. Robach and G. Saucier, "Microprocessor Functional Testing", 1980 Test Conference, Nov. 1980, pp. 433-443.

[10] M.A. Annaratone and M.G. Sami, "An Approach to Functional Testing of Microprocessors", 12th International Symposium on Fault-Tolerant Computing, Santa Monica, CA, June 1982, pp. 158-164.

[11] B. Courtois, "Functional Testing of the Control Section of Integrated Processing Units", RR No. 203, IMAG, Univ. of Grenoble, France, May 1980.

[12] F.C. Hennie, Finite-State Models for Logical Machines, John Wiley & Sons, Inc., NY, 1968.

[13] A.D. Friedman and D. Menon, Fault Detection in Digital Circuits, Prentice-Hall, NJ, 1971.

END

FILMED

1-86

DTIC